UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/085,455	02/27/2002	Motohiro Kawahito	JP920000420US1	1801

7590 06/29/2009

ANNE V. DOUGHERTY
3173 CEDAR RD.
YORKTOWN HEIGHTS, NY 10598

EXAMINER	
PHAM, CHRYSTINE	
ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
06/29/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte MOTOHIRO KAWAHITO, HIDEAKI KOMATSU,
and JOHN CRAIG WHALEY

Appeal 2008-003996
Application 10/085,455
Technology Center 2100

Decided:[1] June 29, 2009

Before ALLEN R. MACDONALD, HOWARD B. BLANKENSHIP, and DEBRA K. STEPHENS, *Administrative Patent Judges*.

MACDONALD, *Administrative Patent Judge.*

DECISION ON APPEAL

---

[1] The two-month time period for filing an appeal or commencing a civil action, as recited in 37 C.F.R. § 1.304, begins to run from the decided date shown on this page of the decision. The time period does not run from the Mail Date (paper delivery) or Notification Data (electronic delivery).

Appellants appeal under 35 U.S.C. § 134 from the Examiner's rejection of claims 1-16, which are all the claims remaining in the application. We have jurisdiction under 35 U.S.C. § 6(b).

We affirm.

STATEMENT OF THE CASE

According to Appellants, the invention relates to a method for optimizing a computer program based on the property of data at the time of execution.[2]

Exemplary Claims

1. A program optimization method for translating, into machine code, source code for a program written in a programming language, and for dynamically optimizing said program comprising the steps of:

performing a dynamic analysis during execution to determine whether the execution speed of said program can be increased by fixing, in a specific state, a parameter for a predetermined command in said program; and

employing results of said analysis for the dynamic generation, in said program, of a path along which said parameter of said predetermined command is fixed in said specific state.

5. A program optimization method for translating, into machine code, the source code for a program written in an object-oriented

[2] Spec. 1:5-8.

programming language, and for optimizing said program comprising the steps of:

detecting one command dynamically during execution, of the commands in said program, for which a method call destination can be identified, and for which the processing speed can be increased by identifying said method call destination; and

dynamically generating a path wherefor said method call destination for said detected command is limited in order to increase the processing speed of said command.

9. The compiler according to claim 8, wherein, in accordance with the state of said program at execution, said specialization unit generates, in said program, a branching process for selectively performing a specialized path and an unspecialized path; and wherein, while taking into account a delay due to the insertion of said branching process, said data specialization selector determines a state in which said parameter of said predetermined command is fixed.

Prior Art

The Examiner relies on the following prior art references to show unpatentability:

Linden	2002/0066086 A1	May 30, 2002
Shaylor	6,760,907 B2	Jul. 6, 2004

Examiner's Rejections

1. The Examiner rejected claims 1-4 and 6-16 under 35 U.S.C. § 102(e) as being anticipated by Linden.

2. The Examiner rejected claim 5 under 35 U.S.C. § 103(a) as unpatentable over Linden and Shaylor.

102(e) REJECTION OVER LINDEN

ISSUE

The issue before us is whether the prior art teaches (1) fixing a parameter for a predetermined command in a specific state and (2) dynamically generating a path along which that parameter is fixed in that state.

FINDINGS OF FACT

1. The Examiner found "Linden explicitly discloses the dynamic decoding (i.e., analysis) stage going through block of source instructions and analyzes the operation codes such as the addition (i.e., command) of two numbers (i.e., parameters)" (Ans. 16) (emphasis omitted).

2A. We also find Linden discloses the dynamic compiler 10 comprises a decoding stage for decoding the source instructions and parameters and for creating an instruction stream that is optimized based on the source instructions and parameters (*see* Linden, ¶ [0034]).

2B. We also find that Linden discloses "at the decoding stage 12, the source instructions are fetched at 40" (Linden, ¶ [0037]) (emphasis omitted).

2C. Further, we find Linden discloses "[t]he decoding stage goes through the block of source instructions and analyzes, examines the operation codes, and outputs an instruction string which is basically a breakdown of what the functional operations are doing" (Linden, ¶ [0037]).

2D. In addition, we find that Linden discloses "[t]aking the decoded instruction, a preliminary optimization of the instruction stream is conducted at 44 to include . . . optimization tasks" (Linden, ¶ [0038]) (emphasis omitted).

3. One of the optimization tasks disclosed in Linden, as found by the Examiner, involves "translating the assignment (i.e., command) of Register 3 (i.e., one of the parameters) to the outcome of 'Register 1 XOR Register 1' which is always fixed in a specific state because clearly it does not matter what the value of variable Register 1 is, the result of XOR-ing a variable and itself is always the constant 0 (i.e., Boolean false)" (Ans. 16, emphasis omitted; *see also* Linden, ¶ [0038]).

4. Further, we find that Linden discloses that "[t]he preliminary optimization step 44 outputs the optimized instruction stream in preparation for subsequent flow optimization and encoding." (Linden, ¶ [0038]) (emphasis omitted).

PRINCIPLES OF LAW

In rejecting claims under 35 U.S.C. § 102, "[a] single prior art reference that discloses, either expressly or inherently, each limitation of a claim invalidates that claim by anticipation." *Perricone v. Medicis*

Pharmaceutical Corp., 432 F.3d 1368, 1375 (Fed. Cir. 2005) (citing *Minn. Mining & Mfg. Co. v. Johnson & Johnson Orthopaedics, Inc.*, 976 F.2d 1559, 1565 (Fed. Cir. 1992)).

Analysis of whether a claim is patentable over the prior art under 35 U.S.C. § 102 begins with a determination of the scope of the claim. We determine the scope of the claims in patent applications not solely on the basis of the claim language, but upon giving claims their broadest reasonable construction in light of the specification as it would be interpreted by one of ordinary skill in the art. *In re Am. Acad. of Sci. Tech Ctr.*, 367 F.3d 1359, 1364 (Fed. Cir. 2004). The properly interpreted claim must then be compared with the prior art.

If the Examiner's burden is met, the burden then shifts to the Appellants to overcome the prima facie case with argument and/or evidence. Obviousness is then determined on the basis of the evidence as a whole and the relative persuasiveness of the arguments. *See In re Oetiker*, 977 F.2d 1443, 1445 (Fed. Cir. 1992).

ANALYSIS

Appellants' Arguments and Contentions

Claims 1, 6, 7, 10, 13, and 15

Exemplary claim 1 recites (1) "determine whether the execution speed of said program can be increased by fixing, in a specific state, a parameter for a predetermined command in said program," and (2) "dynamic generation, in said program, of a path along which said parameter of said predetermined command is fixed in said specific state" (App. Br. 27, Claims Appendix).

Fixing a Parameter for a Pre-determined Command

Appellants contend "Linden creates new instructions for the target device 'independent of operations specified by the source instructions' whereas the present invention uses the 'predetermined command in said program' but dynamically generates a path along which a parameter for the command is fixed in a specific state" (App. Br. 13). Further, Appellants contend "Linden does not look at a parameter of a predetermined command; rather, Linden looks [at the] intent and purpose of instructions" (App. Br. 13-14).

In addition, Appellants argue (1) "that eliminating operations is not the same as the claimed step of determining whether execution speed can be increased by fixing a parameter for a command of the program in a specific state" (App. Br. 14); (2) "Linden's step for making a result a constant does not anticipate fixing a variable parameter" (App. Br. 15); and (3) "Linden does not state that any parameters be fixed" (App. Br. 16).

We find that Linden discloses Register 3 = Register 1 XOR Register 1 (*see* FF 3). Thus, we find that Linden discloses the assignment command, as a skilled artisan would have recognized that the "=" after Register 3 is an instruction to assign what follows the "=" to Register 3 (*see* FF 3).

Also, we agree with the Examiner that "Register 3" is one of the parameters. Further, we also agree with the Examiner that Register 3 is always fixed in a specific state because (1) Register 3 = Register 1 XOR Register 1, and (2) when you XOR something with itself, you always get a false or "0" (*see* FF 3). Thus, we find one of ordinary skill in the art would have recognized that Linden discloses a parameter (*i.e.*, Register 3) fixed

(*i.e.*, set at "0") for a predetermined command (*i.e.*, the assignment command, as represented by the "=").

Performing a Dynamic Analysis and Dynamically Generating a Path Along Which the Parameter of a Predetermined Command is Fixed

Appellants contend "the Examiner has not cited any teaching in Linden which anticipates the claim language of 'performing a dynamic analysis'" (App. Br. 15). Appellants argue "[t]he Examiner has listed terms including 'dynamically cross-compiling, execution speed, execution time, overhead . . . dynamic recompiliation . . . decoded instruction, instruction sequence, result, constant, Register3=0,'" and (2) "[s]imilarly, with respect to the claim language of 'employing results . . .' the Examiner has listed terms (e.g., 'instruction sequence, result, constant, Register=3, optimization step 44, optimized instruction stream' on page 5-6 of the Final Office Action), but the Examiner does not explain how Linden's use of those terms anticipates the claim language" (App. Br. 15). Further, Appellants contend that "the Examiner's duty [is] to apply teachings of the cited art to the claims in order to establish anticipation, and not to simply list terms from the reference" (App. Br. 15).

Appellants also contend "Linden does not dynamically create a path along which a parameter of a predetermined command is fixed in a specific state" (App. Br. 14). Appellants argue that "Linden teaches translating an instruction sequence into another sequence where the result is always a constant (paragraph [[0038]]), but that is not the same as or suggestive of dynamically creating a path along which a parameter of a predetermined command is fixed in a specific state" (App. Br. 14). Appellants' arguments

with regard to exemplary claim 1, also apply to independent claims 7, 10, 13, and 15 (*see* App. Br. 12, 13 and 17).

We agree with the Examiner that Linden discloses a decoding stage that analyzes operation codes (*see* FF 1). Further, we find a skilled artisan would have recognized that Linden's *dynamic* compiler would have performed the analysis of the operation codes (*see* FF 2A). Accordingly, we find Linden discloses performing a dynamic analysis.

In addition, we find that Linden discloses that "[t]he preliminary optimization step 44 outputs the optimized instruction stream in preparation for subsequent flow optimization and encoding," and this instruction stream corresponds to the path of exemplary claim 1 (*see* FF 4). Further, we find that the optimized instruction stream results from the dynamic analysis (as discussed above) that included, for example, the translation of the instruction sequence into another sequence where the result is always zero (*see* FF 3).

Accordingly, we find that a skilled artisan would have recognized that the path is generated from step 44 that involved setting "Register 3=0", *i.e.*, fixing a parameter for a predetermined command in said program (*see* FF 3-4). Thus, we find Linden discloses dynamically generating a path along which a parameter is fixed, and thus, Appellants have not persuaded us of error in the Examiner's finding of anticipation regarding claims 1, 7, 10, 13, and 15. Accordingly, we sustain the Examiner's rejection of claims 1, 7, 10, 13, and 15.

Further, (1) we find claim 6 is commensurate in scope with exemplary claim 1, as claim 6 recites "detecting dynamically . . . one command . . . for which a variable can be limited to a predetermined value . . . and generating

a path along which said constant value for said variable of said detected command is fixed," and (2) Appellants arguments with regard to claim 6 are commensurate in scope with Appellants arguments with respect to exemplary claim 1. Thus, for the reasons discussed above with respect to exemplary claim 1, Appellants have not persuaded us of error in the Examiner's finding of anticipation regarding claim 6, and the Examiner's rejection of claim 6 is sustained.

Claims 2, 3-4, 8, 11-12, 14, and 16

ISSUE

The issue before us is whether the prior art teaches (1) obtaining statistical data for the appearance frequency of each available state, (2) and a data specialization selector.

ADDITIONAL FINDINGS OF FACT

5A. The Examiner found support for Linden's teaching or suggestion of "obtaining statistical data for the appearance frequency of each available state . . . to dynamically generate the path" at least in part, in paragraph 0012 of Linden (Ans. 19).

5B. Further, the Examiner found

> Linden explicitly discloses improving the [overall] execution speed by cross-compiling (also referred to as recompilation) programs containing execution loops of instructions that are repeatedly executed hundreds, thousands of times (i.e., appearance frequency of each available state) during execution of the program. The same paragraph also explicitly discloses analyzing and decoding these frequently executed instructions once to generate the target instruction in order to avoid repeated interpretation of the instructions in such loops, thereby

> saving (i.e., increasing) execution time/speed of the translated/compiled code.

(Ans. 19) (emphasis omitted).

5C. The Examiner then found

> it is inherent in Linden that the frequency of being executed for each command is determined (i.e., obtain statistical data for the appearance frequency of each available state) in order to identify which command to dynamically recompile (as opposed to repeatedly interpreting the frequently executed command which would slow down the overall execution speed of the program) to generate the optimized instruction stream (i.e., path) for the target machine.

(Ans. 19-20).

6. We find that Linden discloses:

> The primary reason that overall execution speed is improved by cross-compilation is that most programs contain execution loops of instructions that are repeatedly executed hundreds, thousands, or even millions of times during a typical execution of the program. The source instructions are analyzed and decoded only once, i.e., the first time they are addressed, and the target instruction stream is generated and stored in memory, usually a RAM memory that may be cache memory. By avoiding repeated interpretation of the instructions in such loops, substantial execution time is saved. Consequently, subsequent emulation of the same source instruction may be performed quickly because the decoding overhead is nonexistent.

(Linden, ¶ [0012]).

ANALYSIS

Appellants argue "the Examiner has not cited any teaching in Linden which anticipates the claim language of 'obtaining statistical data for the appearance frequency of each available state'" and for employing the obtained statistical data to dynamically generate the path"

(*see* App Br. 18-19).

The Appellants rely on the arguments set forth above with regard to the teachings of Linden as they related to claims 1, 7, 10, 13, and 15. (App. Br. 18). Appellants also argue "the Examiner lists terms used in the Linden reference (e.g., 'instruction sequence, result, constant, Register=3, optimization step 44, optimized instruction stream' on page 6 of the Final Office Action), but the Examiner does not explain how Linden's use of those terms anticipates the claim language" (App. Br. 18-19).

Appellants also argue "[s]imilarly, there is no teaching or suggestion of a data specialization selector as set forth in Claims 8 and 9" (App. Br. 19). Further, Appellants argue that "the Examiner has a duty to apply teachings of the cited art to the claims in order to establish anticipation, and not to simply list terms from the reference" (App Br. 19).

We find that Linden discloses that the reason the overall execution speed is improved by cross-compilation is that most programs contain execution loops of instructions that are repeatedly executed hundreds, thousands, or even millions of times during a typical execution of the program (*see* FF 5-6). We find that a skilled artisan would have recognized that the execution speed would not have improved if the task of cross-compilation had not been performed to reduce the number of loops of instructions that were repeated. Further, we find that the task of identifying the instructions loops that were repeated, in Linden, inherently teaches or suggests obtaining statistical data for the appearance frequency of each available state (*i.e.* instruction loop or set of commands), as recited in representative claim 2 (*see* FF 5-6).

Further, we find that Linden discloses (1) "[t]he source instructions are analyzed and decoded . . . and stored in memory," and (2) "[b]y avoiding repeated interpretation of the instructions in such loops, substantial execution time is saved" (*see* FF 6). Thus, we also find that a skilled artisan would have recognized that what is stored in storage is the analyzed and decoded source instructions, *i.e.*, the new path, as what is stored in storage avoids repeated interpretation of instructions (inherently as a result of obtaining statistical data for the appearance frequency of each instruction loop or set of commands). Accordingly, we find Linden inherently teaches or suggests employing the obtained statistical data (*i.e.*, using the results of the analysis) to dynamically generate the path), as recited in representative claim 2.

Claim 8 recites that the data specialization selector is "for, when said program is executed, obtaining statistical data for the appearance frequency of each state obtained by said impact analysis unit, and for determining the state in which said parameter of said predetermined command is to be set" (App. Br. 31). As discussed above, Linden teaches or suggests obtaining statistical data for the appearance frequency of each state, and setting a predetermined command. Accordingly, we find that Linden inherently teaches a data specialization selector, as there must inherently be some component that performs those tasks.

Appellants argue claims 2, 3-4, 8, 11-12, 14, and 16 as a group (App. Br. 18, 20). However, Appellants have not persuaded us of error in the Examiner's finding of anticipation regarding claims 2, 3-4, 8, 11-12, 14, and 16. Accordingly, we sustain the Examiner's rejection of claims 2, 3-4, 8, 11-12, 14, and 16.

Claim 9

ISSUE

The issue before us is whether the prior art teaches a specialization unit that generates a branching process for selectively performing a specialized path and an unspecialized path.

ADDITIONAL FINDINGS OF FACT

7. The Specification of the present application discloses "[a]ccording to the invention, the program is so optimized that on a path (a specialized path) along which the parameter of the predetermined command is fixed and on a path (an unspecialized path) along which this parameter is not fixed in the specific stated is selectively executed during the branching process" (Spec. 4).

8. The Examiner found Linden discloses "creating an optimized instruction flow stream which omits the instructions corresponding to the overridden C=A+B operation (i.e., branching process for selectively performing a specialized path) as opposed to removing the C=A+B instruction at the decoding stage" (Ans. 20) (emphasis omitted).

ANALYSIS

Claim 9 depends from claim 8 and further recites

> wherein, in accordance with the state of said program at execution, said specialization unit generates, in said program, a branching process for selectively performing a specialized path and an unspecialized path; and wherein, while taking into account a delay due to the insertion of the branching process, said data specialization

> selector determines a state in which the parameter of the predetermined command is fixed.

(App. Br. 31, Claims Appendix).

Appellants argue

> the instructions listed by the Examiner on page 10 of the Final Office Action, including 'optimized instruction flow stream, optimization rules, pipeline delay[,]' do not provide teachings which anticipate a specialization unit which generates a branching process for selectively performing both a specialized path and an unspecialized path, or a data specialization selector for determining a state in which the parameter of a predetermined command is fixed.

(App. Br. 21) (emphasis omitted).

We find the Specification of the present application defines (1) a specialized path as a path along which the parameter of the predetermined command is fixed, and (2) an unspecialized path as a path along which the parameter is not fixed in the specific state (FF 7). We find that the act of overriding the instruction, C=A+B, corresponds to a branching process, as the path presents two options: (1) a path that includes the C=A+B instruction, as the instruction was not omitted, and (2) a path that overrides the C=A+B instruction (*see* FF 8).

Accordingly, Linden inherently discloses a specialization unit, as a skilled artisan would have recognized that there must be a component to perform the branching process. Thus, Appellants have not persuaded us of error in the Examiner's finding of anticipation regarding claim 9. Accordingly, we sustain the Examiner's rejection of claim 9.

103(a) REJECTION OVER LINDEN AND SHAYLOR

Claim 5

ISSUE

The issue before us is whether the prior art teaches or suggests detecting a command for which a method call destination can be identified and for which the processing speed can be increased by identifying the method call destination.

FINDINGS OF FACT

9. The Examiner found

> that Shaylor's target method should be understood as the method being called (i.e., method call destination) from the calling method. Furthermore, in order to inline the target method into the calling method (i.e., copy the code of the target method into the cod[e] of the calling method), it is inherent that the actual calling command/instruction (inside the calling method) be detected so as to determine the exact position for inlining (i.e., inserting the code of the target method).

(Ans. 22) (emphasis omitted).

10. We find that Shaylor discloses

> [a] common program occurrence is that a first method (the caller method) calls a second method (the target method) in a process referred to as a 'method call'. This method call process is advantageous from a programmer's perspective, but consumes many clock cycles. An important optimization in the compilation of object oriented program code is called 'inlining.['] In fully compiled programs, inlining makes these target method calls more efficient by copying the code of the target method into the calling method.

(Shaylor, col. 2, ll. 48-56).

PRINCIPLES OF LAW

In rejecting claims under 35 U.S.C. § 103, it is incumbent upon the Examiner to establish a factual basis to support the legal conclusion of obviousness. *See In re Fine*, 837 F.2d 1071, 1073 (Fed. Cir. 1988). In so doing, the Examiner must make the factual determinations set forth in *Graham v. John Deere Co.*, 383 U.S. 1, 17 (1966). Discussing the question of obviousness of claimed subject matter involving a combination of known elements, *KSR Int'l v. Teleflex, Inc.*, 550 U.S. 398 (2007), explains:

> When a work is available in one field of endeavor, design incentives and other market forces can prompt variations of it, either in the same field or a different one. If a person of ordinary skill can implement a predictable variation, § 103 likely bars its patentability. For the same reason, if a technique has been used to improve one device, and a person of ordinary skill in the art would recognize that it would improve similar devices in the same way, using the technique is obvious unless its actual application is beyond his or her skill. *Sakraida* [*v. AG Pro, Inc.*, 425 U.S. 273 (1976)] and *Anderson's-Black Rock*[*, Inc. v. Pavement Salvage Co.*, 396 U.S. 57 (1969)] are illustrative—a court must ask whether the improvement is more than the predictable use of prior art elements according to their established functions.

Id. at 417. If the claimed subject matter cannot be fairly characterized as involving the simple substitution of one known element for another or the mere application of a known technique to a piece of prior art ready for the improvement, a holding of obviousness can be based on a showing that "there was an apparent reason to combine the known elements in the fashion claimed." *Id.* at 418. Such a showing requires:

> "some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness" . . . . [H]owever, the analysis need not seek out precise teachings directed to the

> specific subject matter of the challenged claim, for a court can take account of the inferences and creative steps that a person of ordinary skill in the art would employ.

Id. (quoting *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006)).

If the Examiner's burden is met, the burden then shifts to the Appellants to overcome the prima facie case with argument and/or evidence. Obviousness is then determined on the basis of the evidence as a whole and the relative persuasiveness of the arguments. *See In re Oetiker*, 977 F.2d at 1445.

ANALYSIS

Appellants rely on the discussion and arguments presented above with regard to the teachings of Linden (*see* App. Br. 24). Applicants further argue that "the Shaylor reference does not supply those teachings which are missing from Linden" (App. Br. 24).

Appellants argue (1) Shaylor "teaches that the overhead of method calls can be reduced by 'inlining' which copies the code of a target method into a calling method," and (2) "[c]opying the code of the target method into a calling method is not the same as or suggestive of detecting a command for which a method call destination can be identified and for which the processing speed can be increased by identifying the method call destination" (App. Br. 24).

In addition, Appellants contend Shaylor discloses "inlining causes code size to increase, which effectively teaches away from the claimed steps for increasing processing speed for a command" (App. Br. 25). Further, Appellants argue "the alternative direct method call provided in the Shaylor

passage is not the same as or suggestive of dynamically generating a path where a call destination is limited to increase processing speed"
(App. Br. 25).

We agree with the Examiner, that in order to inline the target method into the calling method, it is inherent that the actual calling command/instruction be detected, so as to determine the exact position for inlining (*see* FF 9). Further, we find that, inherently, inlining increases the speed, as Linden discloses inlining is (1) for the purpose of optimization, and (2) makes target method calls more efficient (*see* FF 10). Thus, we find Shaylor does not teach away from increasing the processing speed for a command.

Thus, Appellants have not persuaded us of error in the Examiner's conclusion of obviousness for claim 5, and the rejection of claim 5 is sustained.

DECISION

The Examiner's decision rejecting claims 1-16 is affirmed.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 C.F.R. § 1.136(a)(1)(iv).

AFFIRMED

msc

ANNE V. DOUGHERTY
3173 CEDAR RD.
YORKTOWN HEIGHTS, NY 10598